Centro Universitario de Ciencias Exactas e Ingenierías.

Ciencias de la computación

Seminario de Uso, Adaptación y Explotación de Sistemas Operativos

Becerra Velázquez Violeta del Rocío

Hernandez Lomelí Diego Armando

219750396

INNI- Ingeniería en informática

D02

Actividad de aprendizaje 12

(2.5 Soluciones al interbloqueo, filósofos)

2023/11/7

Tabla de contenido

Tabla de imágenes

Introducción

Se puede definir el interbloqueo como el bloqueo permanente de un conjunto de procesos que compiten por recursos o se comunican entre sí. Este problema afecta a procesos concurrentes que utilizan recursos en un sistema. Además de que todos los interbloqueos surgen de necesidades que no pueden ser satisfechas.

Para que exista interbloqueo es necesario que existan las condiciones de Coffman, las cuales deben cumplirse simultáneamente, por lo que esta actividad centra su enfoque en las posibles soluciones por software a uno de los problemas más famosos de interbloqueo, problema de los filósofos comensales.

Objetivo

El estudiante propondrá una solución al interbloqueo al problema de los filósofos comensales implementando algoritmos de programación con lenguajes de alto nivel.

Instrucciones.

a). Investigue en que consiste el concepto de interbloqueo, también conocido como bloqueo mutuo o abrazo mortal. Así mismo, busque las condiciones que favorecen al interbloqueo y como pueden ser tratadas

Interbloqueo: Momento en que 2 o más procesos no se pueden ejecutar por qué están a la espera de ejecución de otro de los procesos involucrados o a la liberación de recursos necesarios para su propia ejecución.

El interbloqueo existe cuando se dan las siguientes condiciones:

- **Exclusión mutua:** los procesos exigen exclusividad de los recursos.
- **Condición de espera:** los procesos mantienen la posesión de recursos incluso estando en espera de otros.
- **Condición de no apropiación:** los procesos no pueden arrebatar recursos a otros procesos.
- **Condición de espera circular:** los procesos requieren recursos que están siendo ocupados por el proceso que les procede.

Existen diferentes formas de tratar el interbloqueo, por ejemplo, existen los métodos preventivos que se encargar de ajustar al sistema para evitar que suceda el bloqueo mutuo, aún siendo funcional no aprovechan al máximo los recursos, pero son frecuentemente vistos.

Por otro lado, existen los métodos evitativos se imponen condiciones que tratan de obtener mayor provecho de los recursos, sin embargo, no son 100% efectivas y evita/esquiva el interbloqueo en el momento que va a suceder.

Enfocándonos en la prevención del interbloqueo que es la forma más común de tratar el problema tenemos las siguientes reglas.

- Cada proceso debe pedir los recursos a la vez y no seguirá hasta tenerlos todos.
- Si un proceso necesita recursos que ya están en uso, debe liberarlos y si es necesario, solicitarlos de nuevo junto a los que le hayan faltado
- Ordenamiento lineal a los tipos de recursos en los procesos.

b. El problema de los filósofos comensales ejemplifica el interbloqueo y permite realizar diversas propuestas de solución al mismo. Realice un listado de los requerimientos del planteamiento y cada uno de los inconvenientes involucrados, así como sus posibles soluciones.

El problema de los filósofos comensales tiene a 5 filósofos comensales a su disposición en una mesa redonda con un máximo de 5 cubiertos(tenedores) y 5 platos (1 para cada filósofo). Los filósofos comensales estarán distribuidos a alrededor de toda la mesa en distancias iguales uno del otro con 1 plato enfrente suyo, la distribución de los tenedores es muy similar a la de los filósofos comensales, pero cada tenedor debe quedar a disposición de exactamente 2 filósofos, esto quiere decir que cada personaje debe tener posibilidad de tomar 2 tenedores, 1 a su izquierda y otro a su derecha.

Todos los filósofos pueden hacer transición a 3 estados posibles (durmiendo, pensando, comiendo) siendo estos:

- **Dormido:** el filósofo no realizará ninguna acción mientras los demás estén realizando las acciones necesarias según su estado, cuando haya dormido lo suficiente pasará al estado **pensando**.
- **Pensando:** hará una revisión al estado de los cubiertos a su alrededor, si puede tomar los 2 a la vez, se actualizará su estado a **comiendo,** pero si cualquiera de los 2 tenedores de los que puede disponer están en uso, quedará a la espera de la liberación de los tenedores.
- **Comiendo:** mantendrá a los cubiertos a su alcance ocupados, ningún otro filósofo será capaz de tomarlos hasta que haya terminado de usarlos, cuando haya terminado su uso, los liberará y regresará al estado **dormido**.

Existen condiciones que hacen que los filosofos mueran de inanición en esta implementación.

- **Condición de carrera:** ocurre cuando 2 filosofos quieren tomar el mismo cubierto, solo se lo quedará el que haya solicitado el acceso primero, un filosofo puede quedar perpetuamente en este estado y no conseguir comer si la paralelización de la ejecución no le es favorable
- **Interbloqueo:** todos los filósofos toman el cubierto ya sea todos a su izquierda o derecha pero no pueden tomar el segundo tenedor que necesitan, en esta situación todos se quedan a la espera.

Para evitar la inanición se pueden aplicar diferentes métodos para la evitar la exclusión mutua, en este contexto consideramos a los filósofos como procesos y a los tenedores como recursos que deben ser protegidos.

Como nota adicional, por la cantidad de cubiertos, no pueden haber más de 2 filósofos comiendo a la vez por qué cada uno necesita 2 tenedores y solo disponemos de 5 en total.

c. Una vez verificado en que consiste, deberá programarlo proponiendo una solución a su elección (Monitores, Semáforos, Mutex, Dekker, Peterson, Paso de Mensajes).

Se eligieron los semáforos como solución para el acceso a uso de los tenedores y prevención del interbloqueo y la inanición.

d. El lenguaje de programación a utilizar es a su elección.

Se selecciono **javascript** para la realización de esta actividad con apoyo en **html** y **css** para la interfaz gráfica.

e. Genere un reporte donde incluya la explicación del problema, la solución que propone y la explicación de su código (incluya imágenes).

Se generarán 5 filósofos y 5 tenedores respectivamente, la distribución de los elementos se explica en el inciso b se utilizarán funciones asíncronas para la ejecución de los comensales para simular la ejecución multihilo dentro de js.

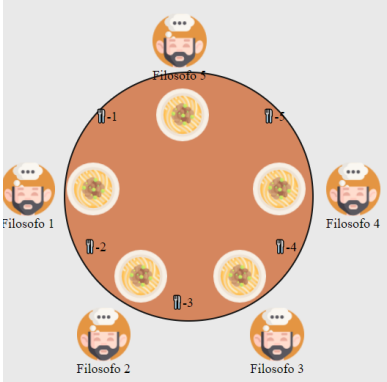

Ilustración 1 Distribución de elementos en la mesa

Los filósofos son marcados con un número dependiendo de su lugar en la mesa y los tenedores también, los tenedores marcados tienen como propósito su identificación y saber en la tarjeta de estados (explicada más adelante) cuales son los adyacentes de cada filosofo

Los filósofos inician estando dormidos y conforme pasa el tiempo despertarán actualizando su estado a pensando, dicho estado los hará analizar los tenedores adyacentes y saber si su semáforo le permite tomarlo, el semáforo de los tenedores es un semáforo numérico entero que solo estará disponible al ser mayor a 0, al ser utilizado su valor disminuirá 1 unidad y al ser liberado aumentará 1 unidad.

Se incluye una tarjeta con los estados existentes y una breve descripción únicamente con fines informativos no funcionales.

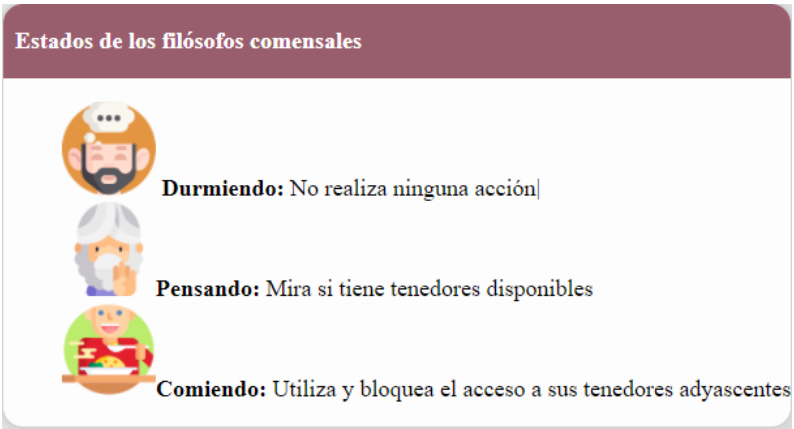

Ilustración 2 descripción de los estados disponibles y su icono

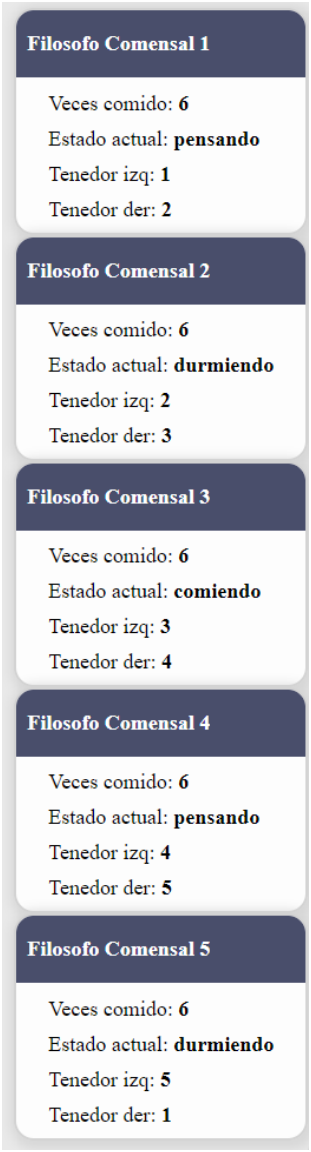

Ilustración 3 tarjetas de estados

Al costado se incluyen tarjetas de estados de cada filosofo comensal, se actualizan cuando el filósofo interactúa, incluye las veces que se ha alimentado y además escribe los tenedores que tiene a su alcance

El resultado de los elementos de la interfaz es el siguiente

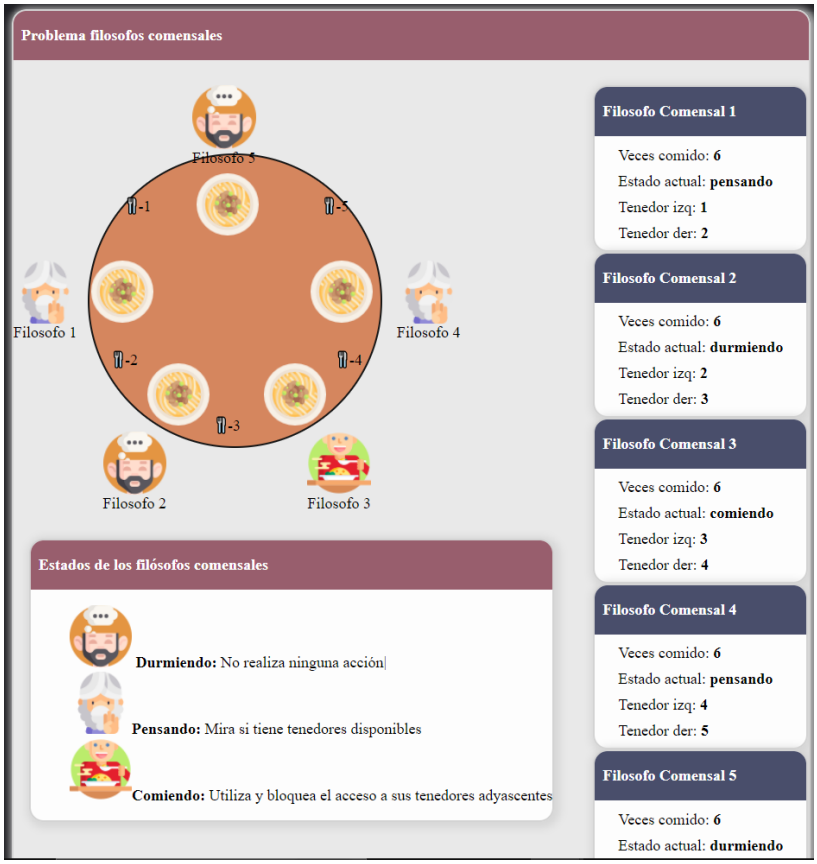

Ilustración 4 interfaz final

La **implementación funcional** del programa se divide en 3 archivos principales:

- Tenedor: define la clase **tenedor** que actúa como clase modelo, es decir, es principalmente para uso de lectura, incorpora en sí un semáforo para controlar su propio uso y es capaz de escribir su índice y el valor del semaforo que tiene.
- FilosofoComensal: contiene la clase **FilosofoComensal** que tiene su estado actual, veces que ha sido alimentado y los índices que corresponden a sus cubiertos adyascentes.

También contiene la clase **GestorComensales** que contiene a todos los comensales y los tenedores disponibles, también se encarga de iniciar el ciclo de ejecución para el acceso de los comensales a los tenedores.

- Ciclo: se encarga de usar las instancias de los archivos anteriores y lanzar el funcionamiento de todo.

Implementación de la clase **tenedor.js**.

```
class Tenedor {
  constructor() {
    this._semaforo = 1;
  }

  usar() {
    if (this._semaforo > 0) {
      this._semaforo--;
      return true;
    }
    this._semaforo--;
    return false; //Se debe bloquear externamente al recurso que quiere usar
  }

  liberar() {
    this._semaforo++;
  }

  get semaforo() {
    return this._semaforo;
  }
  set semaforo(in_semaforo) {
    this._semaforo = in_semaforo;
  }

  get disponible() {
    return this._semaforo > 0;
  }

  escribirSemaforo(index) {
    document.getElementById(`tenedor${index}`).innerText = `🍴-${index + 1}`;
    //-${this._semaforo}
  }
}
```

Implementación de la clase **filosofoComensal.js**.

```
class FilosofoComensal {
  static estados = {
    1: "durmiendo",
    2: "pensando", //Hambriento
    3: "comiendo",
  };
  static imgs = {
    1: "durmiedo.png",
    2: "pensando.png", //Hambriento
    3: "comiendo.png",
  };
  constructor() {
    this._estadoActual = 1;
    this._vecesAlimentado = 0;
    this._izq; //indices en los que tomará al tenedor
    this._der; //indices en los que tomará al tenedor
    this._progresoEstado = 1; //Al contar a 3 se debe cambiar al estado siguiente
  }
  get estadoActual() {
    return this._estadoActual;
  }
  set estadoActual(in_estadoActual) {
    this._estadoActual = in_estadoActual;
  }
  get izq() {
    return this._izq;
  }
  set izq(in_izq) {
    this._izq = in_izq;
  }

  get der() {
    return this._der;
  }
  set der(in_der) {
    this._der = in_der;
  }

  dormir() {
    this._estadoActual = 1;
  }
  hambriento() {
    this._estadoActual = 2;
  }
```

```
  comer() {
    this._estadoActual = 3;
  }
  get progresoEstado() {
    return this._progresoEstado;
  }
  set progresoEstado(in_progresoEstado) {
    this._progresoEstado = in_progresoEstado;
  }
  get imgActual() {
    return FilosofoComensal.imgs[this._estadoActual];
  }

  //Debe ser utilizado al final de la ejecución de las acciones necesarioas
  progresarEstado() {
    if (this._progresoEstado % 3 == 0) {
      //Cambiar de estado
      this._estadoActual = this._estadoActual >= 3 ? 1 : this._estadoActual + 1;
      this._progresoEstado = 1; //Reiiciamos progreso
      return;
    }
    //Avanzar progreso estado
    this._progresoEstado++;
  }

  get vecesAlimentado() {
    return this._vecesAlimentado;
  }
  set vecesAlimentado(in_vecesAlimentado) {
    this._vecesAlimentado = in_vecesAlimentado;
  }
  //en desuso: Inicia el ciclo de vida del filosofo comensal
  despertar() {
    setInterval(() => {
      if (this._estadoActual == 1) {
        //está dormido, no hacer nada
        return;
      }
      if (this._estadoActual == 2) {
        //intenta acceder a los cubiertos
        return;
      }
      //Ya debe estar comiendo en este punto
    }, 100);
  }
```

```
}

class GestorComensales {
  constructor() {
    this.comensales = [];
    this.tenedores = [];
    for (let index = 0; index < 5; index++) {
      let filosofo = new FilosofoComensal();
      filosofo.izq = index;
      filosofo.der = (index + 1) % 5;
      this.comensales.push(filosofo);
      let tenedor = new Tenedor();
      this.tenedores.push(tenedor);
    }
  }

  todosComieron() {
    let todosComieron = true;
    /* console.log(this.comensales); */
    this.comensales.map((comensal) => {
      /* console.log(comensal.vecesAlimentado); */
      if (!todosComieron) {
        //Por lo menos uno no comío, ya no revisamos los demas
        return;
      }
      //Seguirá siendo "true" si ha comido 6 veces
      todosComieron = comensal.vecesAlimentado >= 6;
    });
    /* console.log(todosComieron); */
    return todosComieron;
  }

  despertar() {
    //lanzar función para separar ejecución de todos los filosofos
    this.comensales.map((comensal, index, a) => {
      sleep(10);
      //Se debe ejecutar de manera asincrona
      setInterval(() => {
        document.getElementById(`estadoFilosofo${index}`).src =
          comensal.imgActual;
        document.getElementById(`filosofo${index}`).innerText = `Filosofo ${
          index + 1
        }`;
        //).innerText = `f${index}t i${comensal.izq} d${comensal.der}
c${comensal.vecesAlimentado}`;
```

```
        document.getElementById(
          `vecesComido${index}`
        ).innerText = `${comensal.vecesAlimentado}`;
        document.getElementById(`estado${index}`).innerText = `${
          FilosofoComensal.estados[comensal.estadoActual]
        }`;
        /* Escritura tenedor */
        document.getElementById(`tenedorIzq${index}`).innerText = `${
          comensal.izq + 1
        }`;
        document.getElementById(`tenedorDer${index}`).innerText = `${
          comensal.der + 1
        }`;
        if (this.todosComieron()) {
          return;
        }
        //console.log(comensal.estadoActual);
        this.tenedores[comensal.izq].escribirSemaforo(comensal.izq);
        this.tenedores[comensal.der].escribirSemaforo(comensal.der);

        if (comensal._estadoActual == 1) {
          //está dormido, no hacer nada
          comensal.progresarEstado();
          return;
        }
        if (comensal._estadoActual == 2) {
          //intenta acceder a los cubiertos
          //Si alguno de los tenedores no esta disponible, no avanzamos
          if (
            !this.tenedores[comensal.izq].disponible ||
            !this.tenedores[comensal.der].disponible
          ) {
            sleep(1000); //Detenemos la ejecución para no forzar demasiado el
flujo

            return;
          }
          //Siguen sin ser disponibles

          //Ya están disponibles ambos, se pueden tomar
          this.tenedores[comensal.izq].usar();
          this.tenedores[comensal.der].usar();

          comensal.comer();
          return;
```

```
        }
        //En este punto ya esta comiendo
        if (comensal.progresoEstado % 3 == 0) {
          //Si está apunto de terminar entonces se debe liberar el recurso
          this.tenedores[comensal.izq].liberar();
          this.tenedores[comensal.der].liberar();
          comensal.vecesAlimentado++;
        }
        comensal.progresarEstado();
      }, 500);
    });
  }
}
```

Implementación de **ciclos.js**.

```
function sleep(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

function iniciarCiclo() {
  ciclo();
}

function ciclo() {
  sleep(1000);
  //validar que todos hayan comido
  if (gestorComensales.todosComieron()) {
    console.log("terminado");
    return;
  }
  //continuar ciclo de ejecución
  gestorComensales.despertar();
  //window.requestAnimationFrame(iniciarCiclo);
}

let gestorComensales = null;
document.addEventListener("DOMContentLoaded", (event) => {
  gestorComensales = new GestorComensales();
  ciclo();
});
```

Implementación de **interfaz.html.**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Filosofos comensales</title>
    <script src="ciclo.js"></script>
    <script src="filosofoComensal.js"></script>
    <script src="tenedor.js"></script>
    <style>
      body {
        background-color: #1f1f23;
      }
      .card {
        background-color: rgba(255, 255, 255, 0.9);
        border-color: #fff;
        border: 2px solid lightgray;
        box-shadow: 2px 2px 8px 4px #d3d3d3d1;
        border-radius: 15px;
        overflow: hidden;
      }
      .comensales {
        position: relative;
        margin-right: 20px;
        width: 420px;
        height: 450px;
      }
      .espaciadoMesa {
      }
      .comensales li {
        display: flex;
        flex-direction: column;
      }
      .comensales li:nth-child(1) {
        position: absolute;
        left: 0;
        top: 40%;
      }
      .comensales li:nth-child(2) {
        position: absolute;
        left: 20%;
        bottom: 10px;
      }
      .comensales li:nth-child(3) {
```

```
    position: absolute;
    right: 20%;
    bottom: 10px;
  }
  .comensales li:nth-child(4) {
    position: absolute;
    top: 40%;
    right: 0;
  }
  .comensales li:nth-child(5) {
    position: absolute;
    left: 40%;
  }
  /* ================================================
  Tenedores
  ================================================ */
  .comensales li:nth-child(6) {
    position: absolute;
    left: 25%;
    top: 25%;
  }
  .comensales li:nth-child(7) {
    position: absolute;
    left: 22%;
    bottom: 35%;
  }
  .comensales li:nth-child(8) {
    position: absolute;
    left: 45%;
    bottom: 20%;
  }
  .comensales li:nth-child(9) {
    position: absolute;
    right: 22%;
    bottom: 35%;
  }
  .comensales li:nth-child(10) {
    position: absolute;
    top: 25%;
    right: 25%;
  }
  /* Comida */
  .comensales li:nth-child(11) {
    position: absolute;
    left: 80px;
```

```
    top: 40%;
  }
  .comensales li:nth-child(12) {
    position: absolute;
    left: 30%;
    bottom: 100px;
  }
  .comensales li:nth-child(13) {
    position: absolute;
    right: 30%;
    bottom: 100px;
  }
  .comensales li:nth-child(14) {
    position: absolute;
    top: 40%;
    right: 90px;
  }
  .comensales li:nth-child(15) {
    position: absolute;
    left: 41%;
    top: 90px;
  }

  ul {
    list-style-type: none;
  }
  .mesa {
    border-radius: 50%;
    border: 2px solid rgb(23, 23, 23);
    background-color: rgba(209, 112, 64, 0.82);
    left: 70px;
    top: 140px;
    width: 300px;
    height: 300px;
    position: absolute;
    margin: 1em;
  }
  .spacioFilosofo p {
    margin: 0.5em;
  }
  .spacioFilosofo {
    padding-left: 1em;
    padding-right: 1em;
  }
  .headerProductor {
```

```
      background-color: #494e6b;
      overflow: hidden;
      margin-top: 0px;
      margin-bottom: 10px;
      color: #fff;
      padding-left: 0.5em;
      padding-top: 1em;
      padding-bottom: 1em;
    }
    .headerConsumidor {
      background-color: #985e6d;
      margin-top: 0px;
      margin-bottom: 10px;
      color: #fff;
      padding-left: 0.5em;
      padding-top: 1em;
      padding-bottom: 1em;
    }
    .statesSection {
      margin-left: 1em;
      margin-bottom: 1em;
    }
  </style>
</head>
<body>
  <div class="card" style="width: 100%; height: 100%">
    <h4 class="headerConsumidor">Problema filosofos comensales</h4>
    <div style="width: 100%; height: 100%; display: flex">
      <!-- Comensales -->
      <div class="mesa"></div>
      <div class="espaciadoMesa">
        <ul class="comensales">
          <li>
            <img id="estadoFilosofo0" src="" alt="" /><span
              id="filosofo0"
            ></span>
          </li>
          <li>
            <img id="estadoFilosofo1" src="" alt="" /><span
              id="filosofo1"
            ></span>
          </li>
          <li>
            <img id="estadoFilosofo2" src="" alt="" /><span
              id="filosofo2"
```

```
            ></span>
          </li>
          <li>
            <img id="estadoFilosofo3" src="" alt="" /><span
              id="filosofo3"
            ></span>
          </li>
          <li>
            <img id="estadoFilosofo4" src="" alt="" /><span
              id="filosofo4"
            ></span>
          </li>
          <li id="tenedor0">🍴</li>
          <li id="tenedor1">🍴</li>
          <li id="tenedor2">🍴</li>
          <li id="tenedor3">🍴</li>
          <li id="tenedor4">🍴</li>
          <!-- Comida -->
          <li><img src="spagueti 2.png" alt="" /></li>
          <li><img src="spagueti 2.png" alt="" /></li>
          <li><img src="spagueti 2.png" alt="" /></li>
          <li><img src="spagueti 2.png" alt="" /></li>
          <li><img src="spagueti 2.png" alt="" /></li>
        </ul>
        <div class="card statesSection">
          <h4 class="headerConsumidor">
            Estados de los filósofos comensales
          </h4>
          <ul class="">
            <li>
              <img src="durmiedo.png" alt="" />
              <span><b>Durmiendo:</b> No realiza ninguna acción| </span>
            </li>
            <li>
              <img src="pensando.png" alt="" /><span
                ><b>Pensando: </b>Mira si tiene tenedores disponibles
              </span>
            </li>
            <li>
              <img src="comiendo.png" alt="" /><span
                ><b>Comiendo: </b>Utiliza y bloquea el acceso a sus tenedores
                adyascentes
              </span>
            </li>
          </ul>
```

```
        </div>
      </div>
      <!-- Tarjetas de estados -->
      <ul>
        <li class="card">
          <h4 class="headerProductor">Filosofo Comensal 1</h4>
          <div class="spacioFilosofo">
            <p>Veces comido: <b id="vecesComido0">1</b></p>
            <p>Estado actual: <b id="estado0">1</b></p>
            <p>Tenedor izq: <b id="tenedorIzq0">1</b></p>
            <p>Tenedor der: <b id="tenedorDer0">1</b></p>
          </div>
        </li>
        <li class="card">
          <h4 class="headerProductor">Filosofo Comensal 2</h4>
          <div class="spacioFilosofo">
            <p>Veces comido: <b id="vecesComido1">1</b></p>
            <p>Estado actual: <b id="estado1">1</b></p>
            <p>Tenedor izq: <b id="tenedorIzq1">1</b></p>
            <p>Tenedor der: <b id="tenedorDer1">1</b></p>
          </div>
        </li>
        <li class="card">
          <h4 class="headerProductor">Filosofo Comensal 3</h4>
          <div class="spacioFilosofo">
            <p>Veces comido: <b id="vecesComido2">1</b></p>
            <p>Estado actual: <b id="estado2">1</b></p>
            <p>Tenedor izq: <b id="tenedorIzq2">1</b></p>
            <p>Tenedor der: <b id="tenedorDer2">1</b></p>
          </div>
        </li>
        <li class="card">
          <h4 class="headerProductor">Filosofo Comensal 4</h4>
          <div class="spacioFilosofo">
            <p>Veces comido: <b id="vecesComido3">1</b></p>
            <p>Estado actual: <b id="estado3">1</b></p>
            <p>Tenedor izq: <b id="tenedorIzq3">1</b></p>
            <p>Tenedor der: <b id="tenedorDer3">1</b></p>
          </div>
        </li>
        <li class="card">
          <h4 class="headerProductor">Filosofo Comensal 5</h4>
          <div class="spacioFilosofo">
            <p>Veces comido: <b id="vecesComido4">1</b></p>
            <p>Estado actual: <b id="estado4">1</b></p>
```

```
            <p>Tenedor izq: <b id="tenedorIzq4">1</b></p>
            <p>Tenedor der: <b id="tenedorDer4">1</b></p>
          </div>
        </li>
      </ul>
    </div>
  </div>
  <!-- Tenedores -->
 </body>
</html>
```

Nota:

El programa terminara cuando todos los filósofos hayan comido al menos 6 veces, que en pantalla cada vez que coman se visualice el número de lo que llevan. Los filósofos deben ir numerados del 1 al 5.

f. Realice un video con la demostración del programa funcionando, entréguelo en algún sitio en la nube y no olvide proporcionar permisos públicos de visualización (si el video no se ve solo tendrá la mitad de los puntos de esta actividad).

g. Incluya el enlace al video en su reporte.
https://drive.google.com/file/d/1FdoktBV9qwtWoPcLxeaNctgCeFrH9zc7/view?usp=sharing

h. Genere una conclusión sobre lo que ha aprendido en esta actividad e inclúyalo en su documento a entregar.

Este problema es una buena demostración del interbloqueo y la inanición de procesos dentro de un sistema operativo, pues entendemos que los filósofos son los procesos, los tenedores los recursos y la comida la ejecución del procedimiento mismo, además de ser una practica que contiene bastantes elementos útiles para entender la sincronización de hilos a través de recursos compartidos y funciones dedicadas a la sincronización.